

# The `phylo4` S4 classes and methods

Ben Bolker & Peter Cowan

September 29, 2008

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Package Overview</b>	<b>2</b>
<b>3</b>	<b>Using the S4 help system</b>	<b>2</b>
<b>4</b>	<b>Trees without data</b>	<b>3</b>
<b>5</b>	<b>Trees with data</b>	<b>6</b>
<b>6</b>	<b>Subsetting</b>	<b>7</b>
<b>7</b>	<b>Tree-walking</b>	<b>8</b>
<b>8</b>	<b>multiPhylo classes</b>	<b>8</b>
<b>9</b>	<b>Examples</b>	<b>8</b>
<b>A</b>	<b>Definitions/slots</b>	<b>8</b>
A.1	phylo4 . . . . .	9
A.2	phylo4d . . . . .	9
A.3	multiphylo4 . . . . .	9
<b>B</b>	<b>Validity checking</b>	<b>9</b>
<b>C</b>	<b>Hacks/backward compatibility</b>	<b>10</b>

## 1 Introduction

This document describes the new `phylo4` S4 classes and methods, which are intended to provide a unifying standard for the representation of phylogenetic trees and comparative data in R. The `phylobase` package was developed to help both end users and package developers by providing a common suite of tools

likely to be shared by all packages designed for phylogenetic analysis, facilities for data and tree manipulation, and standardization of formats.

For *end-users*, standardization will greatly simplify comparing analyses across different packages by easing data portability, as well as reducing the learning curve involved when using new packages. Users will also benefit by having a common repository of useful functions contained within one base package, for example tools for including or excluding subtrees (and associated phenotypic data) or improved tree and data plotting facilities. For *developers*, the **phylobase** package allows programming efforts to be put directly into developing new solutions for new problems (i.e. new phylogenetic methods) rather than re-coding the same base tools that each package requires. It is hoped that standardization will also synergize the efforts of individual developers into a comparative method community (this sounds stupid— please fix), as well as facilitating code validation by providing a repository for benchmark tests.

On a more abstract level, two motivations for the development of this package were better data checking and abstraction of the tree data formats. Currently **phylobase** is capable of checking that data and trees are associated in the proper fashion, and protects users and developers from accidentally reordering one, but not the other. The **phylobase** package also seeks to abstract the data format so that commonly used information (for example, branch length information or the ancestor of a particular node) can be accessed without knowing the underlying data structure (i.e., whether the tree is stored as a matrix, or a list, or a parenthesis-based format). This is achieved through generic **phylobase** functions which retrieve the relevant information from the data structures. The benefits of such abstraction are multiple: (1) *easier access to the relevant information* via a simple function call (this frees both users and developers from learning details of complex data structures), (2) *freedom to optimize data structures in the future without breaking code*. Having the generic functions in place to “translate” between the data structures and the rest of the program code allows program and data structure development to proceed somewhat independently. The alternative is code written for specific data structures, in which modifications to the data structure requires rewriting the entire package code (often exacting too high a price, which results in the persistence of less-optimal data structures). (3) *providing broader access to the range of tools in phylobase*. Developers of specific packages can use these new tools based on S4 objects without knowing the details of S4 programming.

The base **phylo4** class is modeled on the the **phylo** class in **ape**. **phylo4d** and **multiPhylo4** extend the **phylo4** class to include data or multiple trees respectively. In addition to describing the classes and methods this vignette gives examples of how they might be used.

## 2 Package Overview

The **phylobase** package currently implements the following functions and data structures:

- Data structures for storing a single tree and multiple trees: `phylo4` and `multiPhylo4`?
- A data structure for storing a tree with associated tip and node data: `phylo4d`
- A data structure for storing multiple trees with one set of tip data: `multiPhylo4d`
- Functions for reading nexus files into the above data structures
- Functions for converting between the above data structures and `ape` `phylo` objects as well as `ade4` `phylog` objects
- Functions for editing trees and data (i.e., subsetting and replacing)
- Functions for plotting trees and trees with data

### 3 Using the S4 help system

The S4 help system works similarly to the S3 help system with some small differences relating to how S4 methods are written. The `plot()` function is a good example. When we type `?plot` we are provided the help for the default plotting function which expects `x` and `y`. R also provides a way to smartly dispatch the right type of plotting function. In the case of an `ape` `phylo` object (a S3 class object) R evaluates the class of the object and finds the correct functions, so the following works correctly.

```
> library(ape)
> rand_tree <- rcoal(10)
> plot(rand_tree)
```

However, typing `?plot` still takes us to the default `plot` help. We have to type `plot.phylo` to find what we are looking for. This is because S3 generics are simply functions with a dot and the class name added.

The S4 generic system is too complicated to describe here, but doesn't include the same dot notation. As a result `?plot.phylo4` doesn't work, R does, however, find the right plotting function.

```
> library(phylobase)
> rand_p4_tree <- as(rand_tree, "phylo4")
> plot(rand_p4_tree)
```

All fine and good, but how to we find out about all the great features of the `phylobase` plotting function? R has two nifty ways to find it, the first is to simply put a question mark in front of the whole call:

```
> ?plot(rand_p4_tree)
```

R looks at the class of the `rand_p4_tree` object and takes us to the correct help file (note: this only works with S4 objects). The second way is handy if you already know the class of your object, or want to compare to generics for different classes:

```
> method?plot("phylo4")
```

More information about how S4 documentation works can be found in the methods package, by running the following command.

```
> help("Documentation", package = "methods")
```

## 4 Trees without data

You can start with a tree — an object of class `phylo` from the `ape` package (e.g., read in using the `read.tree()` or `read.nexus()` functions), and convert it to a `phylo4` object.

For example, load the raw *Geospiza* data:

```
> data(geospiza_raw)
> names(geospiza_raw)
```

```
[1] "tree" "data"
```

Convert the S3 tree to a S4 `phylo4` object using the `as()` function:

```
> library(phylobase)
> g1 <- as(geospiza_raw$tree, "phylo4")
> g1
```

	label	node	ancestor	branch.length	node.type
1	N01	15	NA	NA	root
2	N02	16	15	0.29744	internal
3	N03	17	16	0.04924	internal
4	N04	18	17	0.06859	internal
5	N05	19	18	0.13404	internal
6	N06	20	19	0.10346	internal
7	N07	21	20	0.03550	internal
8	N08	22	21	0.00917	internal
9	N09	23	22	0.07333	internal
10	N10	24	23	0.05500	internal
11	N11	25	19	0.24479	internal
12	N12	26	25	0.05167	internal
13	N13	27	26	0.01500	internal
14	fuliginosa	1	24	0.05500	tip
15	fortis	2	24	0.05500	tip
16	magnirostris	3	23	0.11000	tip

17	conirostris	4	22	0.18333	tip
18	scandens	5	21	0.19250	tip
19	difficilis	6	20	0.22800	tip
20	pallida	7	25	0.08667	tip
21	parvulus	8	27	0.02000	tip
22	psittacula	9	27	0.02000	tip
23	pauper	10	26	0.03500	tip
24	Platyspiza	11	18	0.46550	tip
25	fusca	12	17	0.53409	tip
26	Pinaroloxias	13	16	0.58333	tip
27	olivacea	14	15	0.88077	tip

Note that the nodes and edges are given default names if the tree contains no node or edge names.

The `summary` method gives a little extra information, including information on branch lengths:

```
> summary(g1)
```

No root edge.

Phylogenetic tree : g1

Number of tips : 14

Number of nodes : 13

Branch lengths:

mean : 0.1764008

variance : 0.04624379

distribution :

Min. 1st Qu. Median 3rd Qu. Max.

0.00917 0.04985 0.08000 0.21910 0.88080

Print tip labels:

```
> labels(g1)
```

[1]	"fuliginosa"	"fortis"	"magnirostris"	"conirostris"	"scandens"
[6]	"difficilis"	"pallida"	"parvulus"	"psittacula"	"pauper"
[11]	"Platyspiza"	"fusca"	"Pinaroloxias"	"olivacea"	

Print internal node labels (R automatically assigns values):

```
> nodeLabels(g1)
```

[1]	"N01"	"N02"	"N03"	"N04"	"N05"	"N06"	"N07"	"N08"	"N09"	"N10"	"N11"	"N12"
[13]	"N13"											

Print edge labels (also automatically assigned):

```

> edgeLabels(g1)

[1] "E16" "E17" "E18" "E19" "E20" "E21" "E22" "E23" "E24" "E1"  "E2"  "E3"
[13] "E4"  "E5"  "E6"  "E25" "E7"  "E26" "E27" "E8"  "E9"  "E10" "E11" "E12"
[25] "E13" "E14"

Is it rooted?

> isRooted(g1)

[1] TRUE

Which node is the root?

> rootNode(g1)

[1] 15

Does it have any polytomies?

> hasPoly(g1)

[1] FALSE

Does it have branch lengths?

> hasEdgeLength(g1)

[1] TRUE

You can modify labels and other aspects of the tree — for example,

> labels(g1) <- tolower(labels(g1))

```

## 5 Trees with data

The `phylo4d` class matches trees with data. (**fixme: need to be able to use ioNCL!**) or combine it with a data frame to make a `phylo4d` (tree-with-data) object.

Now we'll take the *Geospiza* data from `geospiza_raw$data` and merge it with the tree. However, since *G. olivacea* is included in the tree but not in the data set, we will initially run into some trouble:

```

> g2 <- phylo4d(g1, geospiza_raw$data)

gives

Error in check_data(res, ...) :
  Tip data names are a subset of tree tip labels
(missing data names: platyspiza,pinaroloxias,olivacea)
(extra data names: Platyspiza,Pinaroloxias)

```

We have two problems — the first is that we forgot to lowercase the labels on the data to match the tip labels:

```
> gdata <- geospiza_raw$data
> row.names(gdata) <- tolower(row.names(gdata))
```

To deal with the second problem (missing data for *G. olivacea*), we have a few choices. The easiest is to use `missing.tip.data="OK"` to allow R to create the new object:

```
> g2 <- phylo4d(g1, gdata, missing.tip.data = "OK")
```

(setting `missing.tip.data` to `"warn"` would create the new object but print a warning).

Another way to deal with this would be to use `prune()` to drop the offending tip from the tree first:

```
> g1B <- prune(g1, "olivacea")
> phylo4d(g1B, gdata)
```

You can summarize the new object:

```
> summary(g2)
```

No root edge.

Phylogenetic tree : as(object, "phylo4")

```
Number of tips      : 14
Number of nodes     : 13
Branch lengths:
  mean              : 0.1764008
  variance           : 0.04624379
  distribution :
  Min. 1st Qu.  Median 3rd Qu.    Max.
0.00917 0.04985 0.08000 0.21910 0.88080
```

Comparative data:

Tips: data.frame with 14 taxa and 5 variables

wingL	tarsusL	culmenL	beakD
Min. :3.975	Min. :2.807	Min. :1.974	Min. :1.191
1st Qu.:4.189	1st Qu.:2.929	1st Qu.:2.187	1st Qu.:1.941
Median :4.235	Median :2.980	Median :2.311	Median :2.073
Mean :4.236	Mean :2.991	Mean :2.333	Mean :2.083
3rd Qu.:4.265	3rd Qu.:3.039	3rd Qu.:2.430	3rd Qu.:2.347
Max. :4.420	Max. :3.271	Max. :2.725	Max. :2.824

```

NA's      :1.000   NA's      :1.000   NA's      :1.000   NA's      :1.000
  gonysW
Min.      :1.401
1st Qu.   :1.845
Median    :1.962
Mean      :2.014
3rd Qu.   :2.222
Max.      :2.676
NA's      :1.000

```

Object contains no node data.

Or use `tdata()` to extract the data (i.e., `tdata(g2)`). By default, `tdata()` will retrieve tip data, but you can also get internal node data only (`tdata(tree, "node")`) or — if the tip and node data have the same format — all the data combined (`tdata(tree, "allnode")`).

Plotting calls `plot.phylog` from the `ade4` package.

If you want to plot the data (e.g. for checking the input), `plot(tdata(g2))` will create the default plot for the data — in this case, since it is a data frame [this may change in future versions but should remain transparent] this will be a `pairs` plot of the data.

## 6 Subsetting

The `subset` command offers a variety of ways of extracting portions of a `phylo4` or `phylo4d` tree, keeping any tip/node data consistent.

**tips.include** give a vector of tips (names or numbers) to retain

**tips.exclude** give a vector of tips (names or numbers) to drop

**mrca** give a vector of node or tip names or numbers; extract the clade containing these taxa

**node.subtree** give a node (name or number); extract the subtree starting from this node

Different ways to extract the *fuliginosa-scandens* clade:

```

> subset(g2, tips.include = c("fuliginosa", "fortis", "magnirostris",
+   "conirostris", "scandens"))
> subset(g2, node.subtree = "N07")
> subset(g2, mrca = c("scandens", "fortis"))

```

One could drop the clade by doing

```

> subset(g2, tips.exclude = c("fuliginosa", "fortis", "magnirostris",
+   "conirostris", "scandens"))
> subset(g2, tips.exclude = names(descendants(g2, MRCA(g2, c("difficilis",
+   "fortis")))))

```



Another approach is to pick the subtree graphically, by plotting the tree and using `identify`, which returns the identify of the node you click on with the mouse.

```
> plot(g1)
> n1 <- identify(g1)
> subset(g2, node.subtree = n1)
```

## 7 Tree-walking

`getnodes`, `children`, `parent`, `descendants`, `ancestors`, `siblings`, `MRCA` ... generally take a `phylo4` object, a node (specified by number or name) and return a named vector of node numbers.

## 8 multiPhylo classes

## 9 Examples

### A Definitions/slots

This section details the internal structure of the `phylo4`, `multiPhylo4`, `phylo4d`, and `multiPhylo4d` classes. The basic building blocks of these classes are the `phylo4` object and a dataframe. The `phylo4` tree format is largely similar to the one used by `phylo` class in the package `ape` <sup>1</sup>.

#### A.1 phylo4

Like `phylo`, the main components of the `phylo4` class are:

**edge** an  $N \times 2$  matrix of integers, where the first column ...

**edge.length** numeric list of edge lengths (length  $N$  or empty)

**Nnode** integer, number of nodes

**tip.label** character vector of tip labels (required)

**node.label** character vector of node labels (maybe empty)

**root.edge** integer defining root edge (maybe NA)

We have defined basic methods for `phylo4:show`, `print` (copied from `print.phylo` in `ape`), and a variety of accessor functions (see help files). `summary` does not seem to be terribly useful in the context of a “raw” tree, because there is not much to compute: **end users?**

Print method: add information about (ultrametric, scaled, polytomies (zero-length or structural))?

---

<sup>1</sup><http://ape.mpl.ird.fr/>

## A.2 phylo4d

The `phylo4d` class extends `phylo4` with data. Tip data, (internal) node data, and edge data are stored separately, but can be retrieved together or separately with `tdata(x, "tip")` or `tdata(x, "all")`.

edge data can also be included — is this useful/worth keeping?

## A.3 multiphylo4

## B Validity checking

- number of rows of edge matrix ( $N$ ) == length of edge-length vector (if  $> 0$ )
- (number of tip labels)+(nNode)-1 ==  $N$
- data matrix must have row names
- row names must match tip labels (if not, spit out mismatches)

Default node labels:

## C Hacks/backward compatibility

There is a way to hack the `$` operator so that it would provide backward compatibility with code that is extracting internal elements of a `phylo4`. The basic recipe is:

```
> setMethod("$", "phylo4", function(x, name) {  
+   attr(x, name)  
+ })
```

but this has to be hacked slightly to intercept calls to elements that might be missing. For example, `ape` detects whether log-likelihood, root edges, node labels, etc. are missing by testing whether they are `NULL`, whereas missing items are represented in `phylo4` by zero-length vectors in the slots (or `NA` for the root edge) — so we need code like

```
> if (!hasNodeLabels(x)) NULL else x@node.label
```

to handle these cases.